Melt: LaTeX with OCaml

Romain Bardou

GT ProVal
June 11, 2010

LaTeX versus OCaml

LaTeX:

- Beautiful documents
- Lots of macros
- Lots of packages

OCaml:

- Great programming language

Motivations for Document Programming

Macros are good practise

```
\newcommand{\ty}{\tau}
\newcommand{\subst}[3]{#1[#2/#3]}
```

Document-specific environments

LaTeX libraries

Compute results in the paper itself

Science-fiction (or is it?):

- Type your theorems
- Check your proofs

LaTeX as a Programming Language

```
\long\def\@makecaption#1#2{
  \vskip \abovecaptionskip
  \setbox\@tempboxa
    \hbox{{\sf\footnotesize \textbf{#1.} #2}}
  \ifdim \wd\@tempboxa >\hsize
    {\sf\footnotesize \textbf{#1.} #2\par}
  \else
    \hbox to\hsize{\hfil\box\@tempboxa\hfil}
  \fi}
```

OCaml as a Programming Language

Great:

- Typed
- Clear semantics
- Expressive (higher-order iterators, algebraic types...)
- Readable errors
- Nice syntax
- You already use it

But:

- Does not produce documents

Melt

An attempt to combine

- the beauty of LaTeX type-setting
- the expressivity of OCaml

Hello, World!

`hello.mlt`:

```
emit (document "Hello, world!")
```

Compile:

```
melt -pdf hello.mlt
```

Obtain `hello.pdf`:

Hello, world!

Intermediate Files

After Melt pre-processor, `hello.ml`:

```
open Latex;;
open Melt;;
# 1 "../vide.mlt"
emit (document (mode T ((text "tata"))))
```

After compiling and running, `hello.tex`:

```
\documentclass{article}
\begin{document}
  Hello, world!
\end{document}
```

Text, Math and Code Modes

Text mode: `"..."`

```
"Hello, world!"
```

Math mode: `$...$`

```
$3.141592$
```

Code mode (default): `{...}`

```
let x = "some macro" in
"Some text with {x}"
```

Arbitrary Nesting

```
"I know that $1+2={latex_of_int (1+2)}$"
```

Produces:

I know that $1 + 2 = 3$

Example: Recoding Enumerate

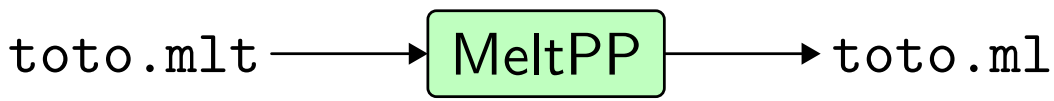
```
let enumerate items =
  let print_item i item =
    "{textbf "{latex_of_int i})"} {item}\\"
  in
  concat (list_mapi print_item items)

...

enumerate ["first"; "second"; "third"]
```

Result:

0) first
1) second
2) third

The Melt Pre-Processor

Provides easy concatenation of text, math and code
(optional) Adds **open** Latex;; **open** Melt;;



The Latex Library

Provides bindings for:

- Many environments
  - `document`, `array`, `itemize`, `figure`, `center`...
- Text type-setting commands
  - `section`, `tableofcontents`, `texttt`, `tiny`, `large`...
- Mathematical symbols
- $\Gamma\rho\epsilon\epsilon\kappa$ letters, hebrew אבגד and $\widetilde{\text{accents}}$
- Beamer
- LaTeX labels and references
- Low-level stuff (`hfill`, `vspace`, `...`)

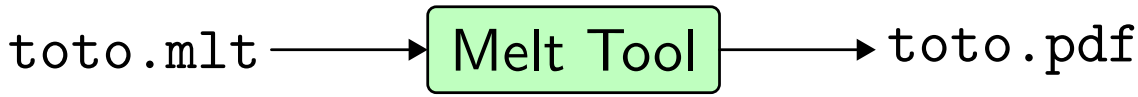
and more.

The Melt Tool

Calls the pre-processor

Compiles, links and executes the OCaml program

Runs `latex` or `pdflatex` and `bibtex`

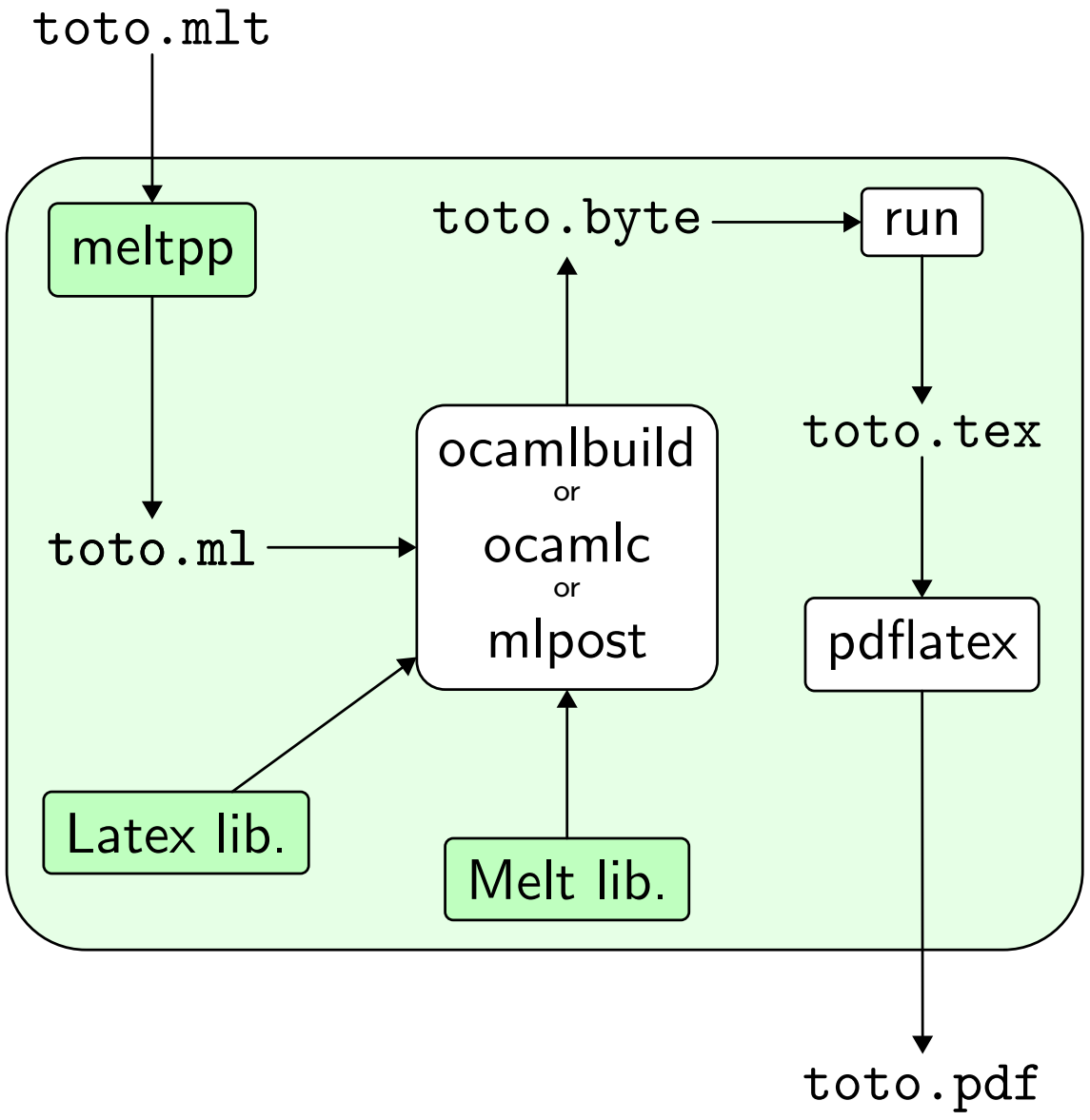


All intermediate files in `_melt` directory

The Melt Library

...and some dirty stuff for the Melt tool

Another Mlpost Diagram



Mlpost Integration

```
val picture_of_latex: Latex.t → Mlpost.Picture.t
val mlpost: Mlpost.Command.t → Latex.t
```

Write your figures in your document:

```
let fancy_text_rotation text =
  let pic = picture_of_latex text in
  ...

let () = emit (document "
  Here is a figure:
  {mlpost (fancy_text_rotation "Text to rotate")}
")
```

Basic Verbatim

Allows to print any symbol.

```
"My webpage: <<http://www.lri.fr/~bardou>>"
```

Generated LaTeX:

```
My webpage: http\symbol{58}\symbol{47}\symbol{47}
www\symbol{46}lri\symbol{46}fr\symbol{47}
\symbol{126}bardou
```

Produces:

My webpage: http://www.lri.fr/~bardou

Much safer than `\verb` or `\begin{verbatim}`.

Pretty-Printed Verbatim

```
let url (x: string) = texttt (Verbatim.verbatim x) in
"My webpage: <:url:<http://www.lri.fr/~bardou>>"
```

Produces:

My webpage: `http://www.lri.fr/~bardou`

In these slides:

- a LaTeX pretty-printer
- an OCaml pretty-printer
- a Melt pretty-printer

Using Verbatim to Ease Writing

A pretty-printer for boolean formulas:

```
let bool =
  Verbatim.pseudocode
    ~symbols: [
      \"/\\\\\", land_;
      \"\\\\/\", lor_;
      \"<=>\", iff;
      \"==>\", rightarrow_;
      \"<==\", leftarrow_;
    ]
    ~keyword_symbols: [\"xor\", oplus; \"xand\", otimes]
```

Using Verbatim to Ease Writing

Let's use our boolean formula pretty-printer:

```
"<:bool:%A /\ B \/ (C_1 xor C_2) <=> (D ==> E_1 xand E_2)%>"
```

Produces:

$$A \wedge B \vee (C_1 \oplus C_2) \iff (D \Rightarrow E_1 \otimes E_2)$$

Motivations for Variables

Collect data following document flow
Use final value before the end

Examples:

- theorem counters
- line numbers in code listings
- titles for a table of contents
- packages used by commands

Variables: Interface

```
type α variable

val variable : α → α variable
val set : α variable → α → t
val get : α variable → (α → t) → t
val final : α variable → (α → t) → t
```

Variables: Example

```
let sections = variable []

let section title =
  concat [
    Latex.section title;
    get sections (fun s → set sections (title :: s));
  ]

let enumerate_sections =
  final sections enumerate
```

Variables: Implementation

Compute a fixpoint on a heterogeneous list of variables
$\Longrightarrow$ a bit tricky

Is it usable in practice?

Yes:

- all of my slides
- all of my research notes
- this very presentation
- the Melt documentation
- several full papers
- several PhD theses

are all written or being written with Melt.

Will it suit your needs?

You won't be stuck with Melt

- you can mix LaTeX and Melt parts
- produced `.tex` files are readable unless lots of verbatim

Several possible programming styles

Based on LaTeX

- use the styles and classes given by your publisher

Try it now!

Webpage:

```
http://melt.forge.ocamlcore.org/
```

Darcs repository:

```
darcs get http://darcs.ocamlcore.org/repos/melt
```

Mailing-list:

```
https://lists.forge.ocamlcore.org/cgi-bin/listinfo/melt-general
```